# EECS 280 Midterm Exam
# Fall 2019
# Multiple Choice and Reference Packet

This is a closed-book exam. You may use one note sheet, 8.5"x11", double--sided, with your name on it. This booklet contains multiple-choice questions, reference code for the written portion, and space for scratch work.

Read the entire exam through before you begin working. Work on those problems you find easiest first. Read each question carefully, and note all that is required of you. Assume all code is in standard C++11, and use only standard C++11 in your solutions.

Instructions:

- **Record your choices for all multiple-choice questions in the written-portion booklet.** Fill in the circle completely. Erase completely if you change your answer
- Throughout the exam, assume all necessary `#include` headers and the `using namespace std;` directive are present unless otherwise directed.
- You do not need to verify `REQUIRES` clauses with `assert` unless instructed to do so.
- The multiple choice booklet **WILL** be turned in. However, **NOTHING** written inside of this booklet will be graded. You must record your answers in the written-portion booklet.
- Turn in both this booklet and the written-portion exam book. One will not be accepted without the other. Keep your note sheet.

## Problem 0: Short Answers (30 Points)

**Record your answers in Question 0 of the written booklet**

### 0a) True/False (10 points)

1. Public class members marked as `static` can be accessed without referencing any particular instance of the class.

2. Consider the following two lines of code:
   ```
   int a = 2;
   _____ x = *(&a);
   ```
   In order to compile, the type of `x` should be (`int *`).

3. Let `ptr_y` be a pointer to a `double`. Then `ptr_y += 1;` will increment the value of `ptr_y` by the size of a `double` instead of `1`.

4. Consider a program with two variables:
   ```
     double b = 3.14;
     double &c = b;
   ```
   These variables correspond to two objects in memory with distinct addresses: one whose value is `3.14`, and the other whose value is the address of `b`.

5. A derived class inherits all public member functions of its base class, including the base class constructor.

6. A derived class has direct access to private member variables of its base class.

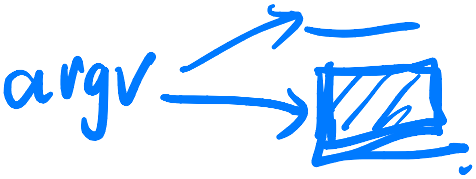
7. Although not all abstract data types (ADTs) are implemented as structs, all structs are examples of ADTs.

8. If `p` is a pointer type, the statement `cout << p;` is guaranteed to print an address.

9. Procedural abstraction involves logically separating what a procedure (or function) does from how it is implemented.

10. The following two declarations:
    ```
      *int const ptr;
      int *const ptr;
    ```
    are functionally identical.

### 0b) Multiple Choice (20 points)



For questions 1-4, consider the following program which has been compiled into an executable named `myprog.exe`:

```
int main(int argc, char **argv){
  cout << *(argv + 1) << endl; //first cout
              argv[1]
  cout << argv[2][4] << endl; //second cout
}
              argv[2]+4
```

Assume the program is then run with the following command:

```
./myprog.exe file1.txt file2.txt 4 1000
```

b 1. What is the output of the first `cout` statement?
   a. `./myprog.exe file1.txt file2.txt 4 1000`
   b. `file1.txt`
   c. `myprog.exe`
   d. `file1.txt file2.txt 4 1000`
   e. `/`

c 2. What is the output of the second `cout` statement?
   a. `0`
   b. `1`
   c. `2`
   d. `2.txt`
   e. `00`

d 3. What is the value of `argc`?
   a. `1`
   b. `2`
   c. `4`
   d. `5`
   e. `1000`

a 4. Which of the following is a valid alternate declaration of argv?
   a. `char *argv[]` ✓
   b. `*char []argv` ✗
   c. `char *argv` ✗
   d. `string *argv` ✗
   e. `argv` definition is implicit if omitted ✗

For questions 5 to 10, consider the following program:

```
void world(char x[], char y[], std::string z){
      if(x == y)
            std::cout << z;
}

int main(){
      char x[] = "hello";
      char y[] = {'h','e','l','l','o'};
      std::string z = "hello";
       // determine behavior at this point
}
```

The program compiles successfully. Determine what would be printed by each of the following statements, assuming they were placed at the end of main. If the statement would cause undefined behavior, select [Undefined]. If nothing is printed, select [Nothing]

d

5. `world(x,y,z);`
   a. z
   b. h
   c. hello
   [✓ d. [Nothing]]
   e. [Undefined]

c

6. `world(y,y,z);`
   a. z
   b. h
   [✓ c. hello]
   d. [Nothing]
   e. [Undefined]

d

7. `world(nullptr, x, z);`

    a. z  
    b. h  
    c. hello  
    [✓ d.] [Nothing]  
    e. [Undefined]

~~b~~ e

8. `cout << y;`

    a. h  
    b. [✓ hello]  
    c. hellohello  
    d. [Nothing]  
    e. [Undefined]

a

9. `cout << *y;`

    [✓ a.] h  
    b. hello  
    c. hellohello  
    d. [Nothing]  
    e. [Undefined]

b

10.
```
for(int i = 0; i < strlen(x); i++)
            cout << x[i];
```

    a. h  
    [✓ b.] hello  
    c. hello\0  
    d. [Nothing]  
    e. [Undefined]

## Problem 1: Strings and I/O (17 Points)

**1a) (9 points)**

Implement the split_string() function according to the RME, along with any other helper functions used

- Your implementation **MUST** use traversal by pointer and NOT traversal by index.
- You **MAY NOT** use any classes or functions from the standard library (e.g. strlen()), but you may implement and call helper functions (do not worry about the relative order that you write the functions, i.e. you may assume all functions are declared before their use)

```
// REQUIRES: src points to a valid C-style string
//           dst points to allocated memory at least as large
//           as allocated for src
//           pos is the position at which to split src and
//           must be less than the length of str
// MODIFIES: *dst
// EFFECTS:  Writes a C-style string (to address specified by
//           dst) which is the concatenation of the substring
//           following and including src[pos], followed by the
//           substring preceding src[pos]
// EXAMPLES:
//        char dst[10];
//         cout << split_string(dst, "hello", 2);   // dst → "llohe"
//         cout << split_string(dst, "racecar", 3); // dst → "ecarrac"
//         cout << split_string(dst, "penny", 0);   // dst → "penny"
//         cout << split_string(dst, "goodbye", 6); // dst → "egoodby"
void split_string(char *dst, const char *src, int pos);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 1A OF THE ANSWER BOOKLET**

**1b) (8 points)** For this problem, you are to implement a program which is called from the command line with an integer N larger than 0, followed by an optional string specifying an input file. If the input file is not specified, input should be read in from cin. Starting from the beginning of the file, the program should read in and print every Nth word (e.g. if N is 3, the 1st word is printed, then the 4th, 7th, etc), to standard output (cout). Each printed word should be followed by a comma. The remaining words are ignored and are not printed.

**Example:** If the program is compiled to `main.exe` and run as:

```
$./main.exe 2 input.txt
```

and “input.txt” consists of the following:

```
She went to the store
```

the following should be printed to standard output:

```
She,to,store,
```

You must write the “main” function for this program (don’t forget to pass in argc and argv). You may write and call any helper functions (do not worry about the relative order that you write the functions, i.e. you may assume all functions are declared before their use). You may use functions from the standard library.

**Error checking:** Your implementation **MUST** check whether there are enough arguments – one integer and an optional filename. If there are not enough, return 1 from `main`. You **DO NOT** need to check for any other errors, such as whether the integer is formatted properly or whether the file was successfully opened. **Remember to handle the case where the input file is not specified.**

**WRITE YOUR IMPLEMENTATION IN QUESTION 1B OF THE ANSWER BOOKLET**

## Problem 2: Arrays and Pointers (18 Points)

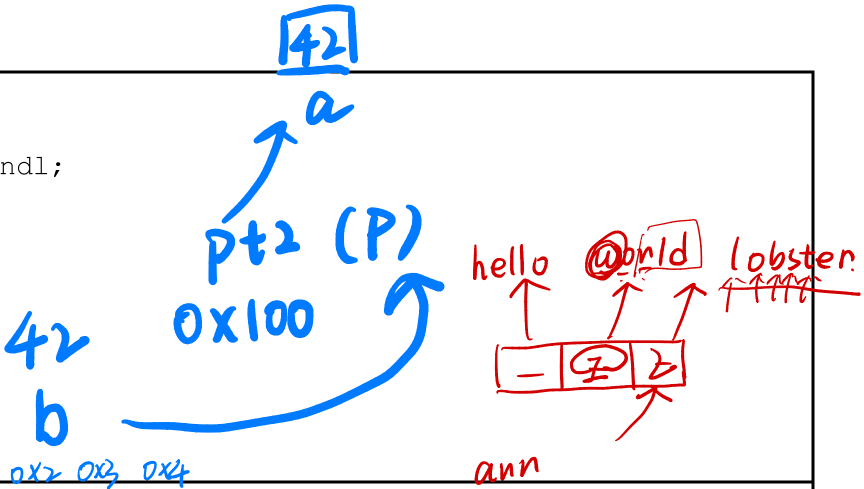
**2a) (8 points)** Write the output of each of the code snippets in the answer booklet. If the code does not compile, write **"compile error"**. If it compiles but results in undefined behavior at runtime, write **"undefined"**. If it compiles and runs but prints nothing, write **"nothing"**. Do not worry about precise whitespace or newlines - we will ignore whitespace when grading.

**1.**

```
void foo(int *a) {
 int *b = a;
 cout << (*b)++ << endl;
}

int main() {
  int a = 42;
  int *ptr = &a;
  int* &p = ptr;
  foo(p);
  return 0;
}
```

42



**2.**

```
int arr[5] = {4, 2, -1, 5, 2};
int *ptr = &arr[2];
ptr = ptr + *ptr;
ptr = ptr - *ptr;
cout << *ptr << endl;
```

0x0 0x1 0x2 0x3 0x4

0x2 *M pt2 -1 2

char **arr.

arr[i] *(arr+1)

**3.**

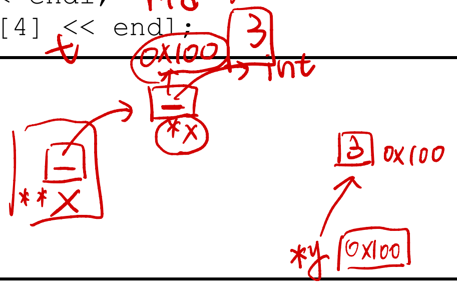
```
const char *arr[] = { "hello", "world", "lobster" };
cout << arr[1] + 2 << endl;
cout << (*(arr + 2))[4] << endl;
```

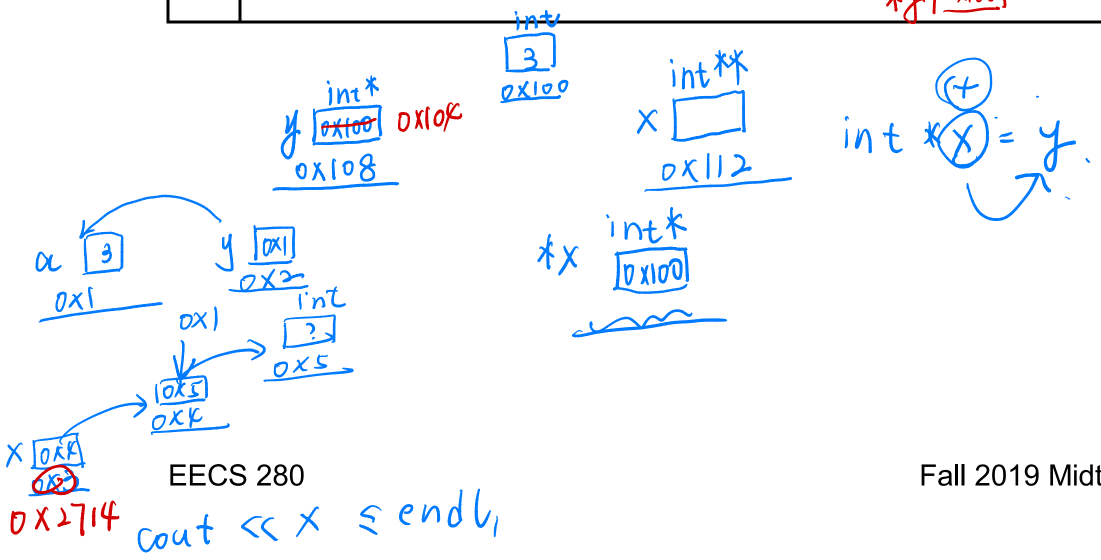
rld

arr[2] t

**4.**

```
int **x;
int *y = 3;
*x = y;
++y;
cout << x << endl;
```





**2b) (3 points)** Write the following function according to its RME.

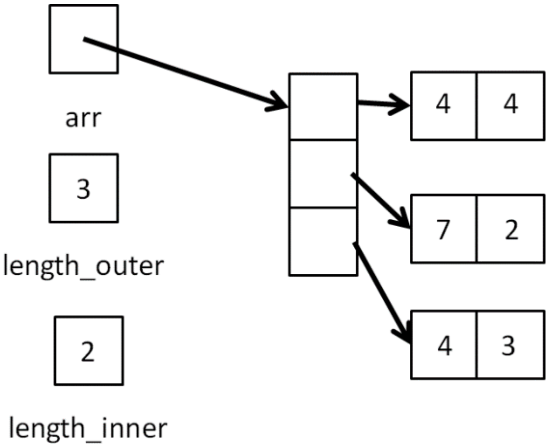
- You may **NOT** call any functions, including those from the standard library.

```
// REQUIRES: length is the length of the array, length >= 1
// EFFECTS:  Returns the range (difference between the maximum
//           and minimum values) of the array
// EXAMPLES: Input: {7, 13, 2, 8, 8, 5, 13}, 7 → (13-2)= 11

int range(const int * array, int length);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 2B OF THE ANSWER BOOKLET**

**2c) (7 points)** Consider a function which takes as input an array of integer arrays, and returns the index of the inner array with the smallest range. In the below example, the function should return 0, since the array {4,4} has the smallest range.



Implement the described function according to its RME.

- Your implementation **MUST** use the **range()** function from **the** previous problem (you may assume it has been implemented correctly).
- You may **NOT** call any other functions, including those from the standard library.

```
// REQUIRES: length_outer is the length of the outer array,
//           length_outer >= 1
//           length_inner >= 1, the length of each inner array
// EFFECTS:  Returns the index of the array with the smallest range.
//           If there is a tie, returns the smaller of the indices
// EXAMPLE:
// Input: {{4, 4}, {7, 2}, {4, 3}}, 3, 2 → 0
int search(int **arr, int length_outer, int length_inner);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 2C OF THE ANSWER BOOKLET**

## Problem 3: Structs and C-Style ADTs (16 points)

Use the following code for Problem 3. We recommend you briefly skim it before answering questions.

The following code defines an abstract data type for a Song:

```
struct Song {
    int num_plays;
    string title;
    string artist;
};

// REQUIRES: s_ptr points to a Song object
// MODIFIES: *s_ptr
// EFFECTS:  Initializes the song to have a given title and artist.
//           All songs are initially not played (i.e. num_plays
//           is zero).
void Song_init(Song *s_ptr, string title_in, string artist_in);

// REQUIRES: s_ptr points to a valid Song
// EFFECTS:  Returns number of times the given song has been played
int Song_plays(const Song *s_ptr);

// REQUIRES: s_ptr points to a valid Song
//           inc is non-negative
// MODIFIES: *s_ptr
// EFFECTS:  Increases the number of times the song has been played
//           by "inc"
void Song_inc_plays(Song *s_ptr, int inc);


// REQUIRES: s_ptr points to a valid Song
// EFFECTS: Returns the title of the given song
string Song_title(const Song *s_ptr);

// REQUIRES: s_ptr points to a valid Song
// EFFECTS: Returns the artist of a given song
string Song_artist(const Song *s_ptr);
```

In this problem, we also define an abstract data type for a Playlist

```
const int MAX_SONGS = 99;

// REQUIRES:
struct Playlist {
    int num_songs;
    string name;
    Song songs[MAX_SONGS];
};

// REQUIRES: p_ptr points to valid playlist,
//           song is a valid Song
// MODIFIES: *p_ptr
// EFFECTS: Appends copy of song to end of playlist,
//          increments num_songs by one
void Playlist_addSong(Playlist *p_ptr, Song song);
```

**3a) (2 points)** Implement the `Playlist_init()`, which is part of the `Playlist` ADT, below according to its RME. You **MUST** respect the interface for the `Song` ADT.

```
// REQUIRES: p_in points to a Playlist object
// MODIFIES: *p_in
// EFFECTS:  Initializes the playlist to have the given name and no
//           songs.
void Playlist_init(Playlist *p_in, string name_in);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 3A OF THE ANSWER BOOKLET**

**3b) (6 points)** Implement the `Playlist_find_song()`, which is part of the `Playlist` ADT, below according to its RME. You **MUST** respect the interface for the `Song` ADT.

```
// REQUIRES: p_in points to a valid Playlist, s_in points to a valid song
//           p_in does not contain duplicate songs
// EFFECTS:  Searches p_in for a Song object which matches both the title
//           and artist. If one exists, returns a pointer to the Song.
//           Otherwise, it returns a null pointer
Song *Playlist_find_song(Playlist *p_in, const Song *s_in);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 3B OF THE ANSWER BOOKLET**

**3c) (8 points)** Implement the `Playlist_merge()`, which is part of the `Playlist` ADT, below according to its RME. You **MUST** respect the interface for the `Song` ADT, and you **MUST** call `Playlist_find_song()in your implementation.`

```
// REQUIRES: p_in1 and p_in2 point to valid Playlists. The sum of the
//           sizes of both playlists is less than MAX_SONGS
// MODIFIES: *p_in1
// EFFECTS:  For any song in *p_in2 that also exists in *p_in1 (i.e. both
//           the artist and song name match), the relevant Song object
//           in *p_in1 is modified so that the number of plays is the sum
//           of the play count of the two Song objects.
//           If the song does not exist in *p_in1, it is added to
//           the end of *p_in1

void Playlist_merge (Playlist *p_in1, const Playlist *p_in2);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 3C OF THE ANSWER BOOKLET**

## Problem 4: Inheritance and Polymorphism (19 Points)

Use the following classes for this problem. We recommend you briefly skim them before answering questions.

```
class Device {
public:
      Device(string brand_in) :
         batteryPercentage(100), brand(brand_in) { }

      virtual void depletePower(int amount) {
            std::cout << "Depleted: " << amount << "%\n";
            batteryPercentage -= amount;
      }

      virtual void startUp() = 0;

      virtual string getDeviceID() = 0;

      string getBrand() {
            return brand;
      }

      int getBatteryPercentage() {
            return batteryPercentage;
      }

private:
      int batteryPercentage;
      string brand;
};
```

```
class Phone : public Device {
public:
      //Constructs a Phone object with the given phone number, brand,
      //and lowPowerMode setting

      Phone(const string &phoneNumber_in, const string &brand_in,
            bool lowPowerModeOn_in);
      //Note: Implementation not shown

      // Depletes the batteryPercentage of the phone by
      // the effective amount: either "amount" if lowPowerMode
      // is off, or "amount/2" otherwise
      //Prints message "Phone Depleted: <effective amount>%"
      //implemented in part A
      virtual void depletePower(int amount);

      void makePhoneCall(const string &contactName) {
            cout << "Calling " << contactName << endl;
      }

      virtual void startUp() {
            cout << "Pinging cell towers!" << endl;
      }

      virtual string getDeviceID() {
            return DeviceID;
      }

      const static string DeviceID = "A123B";


private:
      string phoneNumber;
      bool lowPowerModeOn;
};
```

```
class Laptop : public Device {
public:
      //implemented in part C
      //Constructs a Laptop with the given aspectRatio,
      //brand, and hinge closed
      Laptop(double aspectRatio_in, const string &brand_in);

      void closeHinge() {
            hingeOpen = false;
            cout << "Laptop is now closed" << endl;
      }
      void openHinge() {
            hingeOpen = true;
            cout << "Laptop is now open" << endl;
      }

      virtual string getDeviceID() {
            return DeviceID;
      }

      const static string DeviceID = "1234ABCD";

private:
      double aspectRatio;
      bool hingeOpen;
};
```

**4a) (3 points)** Most modern smartphones have the ability to turn on a low power mode in which the device uses less power by either dimming the display, providing a slower clock update, or limiting other features while still maintaining basic functionality of the phone. Implement the virtual function `depletePower(int amount)` for the "Phone" class such that if low power mode is on, the phone depletes exactly **half** as much as the input "amount" and if low power mode is off, deplete **exactly "**amount" battery percentage. Implement this as it would appear outside of the class definition.

```
void Phone::depletePower(int amount);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 4A OF THE ANSWER BOOKLET**

**4b) (3 points)** Implement a function which returns the number of devices in the vector "devices" that satisfy the following constraints:

- The device has batteryPercentage > 75
- The device has DeviceID equal to that of a "Phone" object.

To receive full credit, your implementation should not “hardcode” the string literal for a phone, but instead read the static data member `DeviceID` from the “Phone” class.

```
int countChargedPhones(const std::vector<Device *> &devices);
```

**WRITE YOUR IMPLEMENTATION IN QUESTION 4B OF THE ANSWER BOOKLET**

**4c) (3 points)** Implement the constructor for the Laptop, as it would appear outside of the class definition. The laptop should be constructed initially with the hinge closed. You must use a member initializer list.

**WRITE YOUR IMPLEMENTATION IN QUESTION 4C OF THE ANSWER BOOKLET**

**4d) (8 points)** Write the output of each of the code snippets in the answer booklet. If the code does not compile, write "compile error". If it compiles but results in undefined behavior at runtime, write "undefined". If it compiles and runs but prints nothing, write "nothing".

| | |
|---|---|
| **1.** | Phone iphone("911", "Apple", true);<br>Device *d_ptr = &iphone;<br>d_ptr->depletePower(4);<br>iphone.depletePower(2); |
| **2.** | Phone galaxy("800-588-2300", "Samsung", false);<br>Device *d_ptr = &galaxy;<br>d_ptr->makePhoneCall(); |
| **3.** | Phone nokia("800-123-4567", "Nokia", false);<br>nokia.startUp(); |
| **4.** | Device blackberry("Blackberry");<br>Phone *p_ptr  = &blackberry;<br>p_ptr->depletePower(10); |

**4e) (2 points)** The following code does not compile. In the answer booklet, briefly (in 20 words or fewer), explain why.

Laptop chromeBook(10, "Google");
chromeBook.openHinge();

## Space for Scratch Work

You may use this space for scratch work, but we will not grade anything here.

## Space for Scratch Work

You may use this space for scratch work, but we will not grade anything here.

## Space for Scratch Work

You may use this space for scratch work, but we will not grade anything here.